Fundamentos de Aprendizaje Automático 2017/2018

PRÁCTICA DE INICIACIÓN A PYTHON

Objetivo

El objetivo de esta práctica es familiarizarnos con el lenguaje de programación Python, ya que éste será el lenguaje a utilizar a lo largo de las prácticas. Además, se deberá implementar parte de la clase *Datos* que se empleará durante el curso para gestionar los diferentes conjuntos de datos con los que probar los algoritmos de reconocimiento de patrones a implementar.

Preliminares

El lenguaje de programación Python es uno de los lenguajes que está adquiriendo mayor popularidad en el campo del reconocimiento de patrones[1]. Dada su versatilidad y sencillez, es el lenguaje que vamos a utilizar a lo largo de las prácticas.

Durante las prácticas emplearemos el lenguaje Python y los Jupyter Notebooks (anteriormente denominados IPython Notebooks) para presentar los resultados. También puede desarrollarse el código con cualquier IDE (como Spyder) si se considera adecuado. Utilizaremos Anaconda, una distribución totalmente gratuita de Python que incluye más de 300 paquetes como NumPy, para computación científica básica y análisis de datos, y Scikit-learn, paquete con diversos algoritmos de aprendizaje automático. Además, Anaconda incluye también el IDE Spyder y los Jupyter Notebooks, que permiten combinar celdas de texto y código, y serán útiles para combinar las implementaciones de código realizadas y la discusión de los resultados obtenidos.

Anaconda permite instalar Python 2.7 o Python 3.6. Se puede descargar en https://www.continuum.io/.

IMPORTANTE. Para poder usar Anaconda en los laboratorios de prácticas, debe establecerse la variable de entorno PATH como:

```
# export PATH=${PATH}:/opt/anaconda-2.3.0/bin
```

Para arrancar el intérprete de IPython de Anaconda, se debe ejecutar el binario que se encuentra en el directorio de instalación de Anaconda (`/opt/anaconda-2.3.0/`):

```
# /opt/anaconda-2.3.0/bin/ipython
```

Y para arrancar el servidor de Notebooks, éste se debe ejecutar desde el directorio de instalación de la siguiente forma:

```
# cd /opt/anaconda-2.3.0
# ./bin/ipython notebook $HOME
```

Las diapositivas disponibles en *Moodle* junto al enunciado de prácticas proporcionan una introducción a Python (variables y tipos de datos, funciones, secuencias de control, ficheros, clases, módulos, ficheros, paquete NumPy, …).

[1] http://machinelearningmastery.com/best-programming-language-for-machine-learning/

Los métodos de aprendizaje automático permiten construir modelos a partir de un conjunto de datos. En las prácticas, utilizaremos algunos conjuntos de datos del repositorio de la Universidad de California Irvine (UCI)[2]. Los conjuntos de datos que utilizaremos vendrán dados por un fichero de datos (extensión `.data`) y generalmente por un fichero con una descripción de los mismos (extensión `.names`). En caso de ser necesario, cada fichero de datos debe tratarse para que siga la siguiente estructura:

- 1ª Fila: Número de datos del conjunto
- 2ª Fila: Nombres de los atributos. El último atributo corresponderá a la clase en el caso de problemas de aprendizaje supervisado.
- 3ª Fila: Tipos de los atributos: Nominal o Continuo
- Resto de filas: Conjunto de datos, uno por fila y campos separados por comas.

Como ejemplo, en esta práctica se proporcionan en *Moodle* los ficheros correspondientes a los conjuntos de datos ***tic-tac-toe*** (http://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame) y ***statlog*** (http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29).

Actividades

La planificación temporal sugerida y las actividades a llevar a cabo son las siguientes:

- *1º semana*: Leer y comprender la presentación de introducción a Python y realizar los ejemplos y ejercicios que hay a lo largo de la misma, prestando especial atención al uso del paquete NumPy que permitirá trabajar fácilmente con matrices.

- *2ª semana:* Implementar la clase *Datos* para leer los datos del fichero de entrada y almacenar la información necesaria para su posterior uso por los algoritmos de aprendizaje automático y métodos de particionado. Una posible estructura de implementación se comenta posteriormente en este enunciado.

Diseño

Con el objetivo de desarrollar una aplicación lo más flexible y general posible se plantea la siguiente estructura para la clase *Datos*:

```
import numpy as np

class Datos(object):

  TiposDeAtributos=('Continuo','Nominal')
  tipoAtributos=[]
  nombreAtributos=[]
  nominalAtributos=[]
  datos=np.array(())
  # Lista de diccionarios. Uno por cada atributo.
  diccionarios=[]
```

[2] http://archive.ics.uci.edu/ml/

```
  # TODO: procesar el fichero para asignar correctamente las variables
tipoAtributos, nombreAtributos,nominalAtributos, datos y diccionarios
  def __init__(self, nombreFichero):

  # TODO: implementar en la practica 1
  def extraeDatosTrain(idx):
    pass

  def extraeDatosTest(idx):
    pass
```

Se deberá implementar el constructor de la clase (__init__) que recibe como parámetro el nombre del fichero de datos. Las funciones extraeDatosTrain y extraeDatosTest se implementarán en la práctica 1, por lo que no hay que desarrollarlas ahora.

Los atributos de la clase deberán guardar la siguiente información:

- **`TiposDeAtributos`**: esta variable no debe tocarse. Guarda las dos posibles descripciones de los tipos de atributos.
- **`tipoDeAtributos`**: Lista con la misma longitud que el número de atributos del problema (incluyendo la clase) y que contendrá el tipo de atributo de cada variable (Continuo o Nominal). En caso de que el fichero de datos contenga algún tipo de datos que no se corresponda a uno de estos dos tipos, se deberá informar del error. Por ejemplo, se puede lanzar una excepción del tipo `ValueError`.
- **`nombreDeAtributos`**: Lista con la misma longitud que el número de atributos del problema (incluyendo la clase) y que contendrá el nombre de cada variable.
- **`nominalAtributos`**: Lista de valores booleanos con la misma longitud que el número de atributos del problema (incluyendo la clase) que contendrá True en caso de que el atributo sea nominal y False en caso contrario. Esta estructura será útil posteriormente para el uso del paquete scikit-learn.
- **`datos`**: Array bidimensional de NumPy (matriz) que se utilizará para almacenar los datos. El uso de NumPy facilitará el indexado de los datos, así como el uso del paquete scikit-learn.
- **`diccionarios`**: lista de diccionarios con la misma longitud que el número de atributos. La posición i-ésima de la lista contendrá el diccionario asociado al atributo i-ésimo. Los diccionarios solo deben generarse para las variables nominales, para variables continuas, se guardará un diccionario vacío. Para las variables nominales, el diccionario establecerá la relación entre los valores categóricos (claves) y un entero (valor). Para garantizar que este mapeo es consistente para diferentes ficheros y permutaciones de los datos, los enteros deben asignarse en orden lexicográfico de las claves. Así, por ejemplo, en el caso del conjunto de datos tic-tac-toe, donde el primer atributo puede tomar los valores ‘x’, ‘b’, ‘o’, el diccionario correspondiente deberá ser: `{'b': 0, 'o': 1, 'x': 2}`. Una vez obtenidos los diccionarios de cada atributo, las variables categóricas deberán ser codificadas a los valores enteros asignados en el diccionario para su posterior almacenamiento en el array NumPy `datos`.

Esta clase se definirá en el módulo faa (fichero **faa.py**). La plantilla de la clase Datos se puede encontrar en *Moodle*. De esta forma, se pueden instanciar elementos de la clase Datos como sigue:

```
from faa import Datos
dataset=Datos('./ConjuntosDatos/tic-tac-toe.data')
```

Fecha de entrega y entregables

Semana del 25 al 29 de Septiembre de 2017. La entrega debe realizarse antes del comienzo de la clase de prácticas correspondiente. Se deberá entregar un fichero comprimido .zip con nombre **`FAAINTRO_<grupo>_<pareja>.zip`** (ejemplo `FAAINTRO_1461.zip`) y el siguiente contenido:

1. Código Python (**faa.py**) con la implementación de la clase Datos

Esta práctica se evalúa como APTO/NO APTO